VYSOKÁ ŠKOLA BÁŇSKÁ – TECHNICKÁ UNIVERZITA OSTRAVA
EKONOMICKÁ FAKULTA

KATEDRA APLIKOVANÉ INFORMATIKY

Tvorba programu pro vytváření a tisk faktur na platformě Java

Development of a Program for Invoice-making and Printing on the Java Platform

Student: Eduard Maléř

Vedoucí bakalářské práce: Ing. Vítězslav Novák, Ph.D.

Ostrava 2010

VŠB - Technická univerzita Ostrava
Ekonomická fakulta
Katedra aplikované informatiky

Akademický rok 2009/2010

Zadání bakalářské práce

Student: **Eduard Maléř**

Studijní program: B6209 Systémové inženýrství a informatika

Studijní obor: 6209R001 Aplikovaná informatika

Téma: Tvorba programu pro vytváření a tisk faktur na platformě Java

Development of a Program for Invoice-making and Printing on the Java Platform

Zásady pro vypracování:

1. Úvod
2. Popis současného stavu
3. Teoretická východiska
4. Návrh a realizace programu
5. Zhodnocení navrhovaného řešení
6. Závěr

Seznam použité literatury
Seznam zkratek
Prohlášení o využití výsledků bakalářské práce
Přílohy

Seznam doporučené odborné literatury:

BURD, B. *JSP: JavaServer Pages – Podrobný průvodce.* 1. vyd. Praha: Computer Press, 2003. 381 s. ISBN 80-7226-804-X.
CYROŇ, M. *CSS - kaskádové styly : praktický manuál.* 1. vyd. Praha: Grada Publishing, 2006. 340 s. ISBN 80-247-1420-5.
HALL, M. *Java servlety a stránky JSP.* 1. vyd. Praha: Neocortex, 2002. 574 s. ISBN 80-86330-06-0.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Vítězslav Novák, Ph.D.**

Datum zadání: 20.11.2009
Datum odevzdání: 07.05.2010

Ing. Jan Ministr, Ph.D.
vedoucí katedry

prof. Dr. Ing. Dana Dluhošová
děkanka fakulty

Místopřísežně prohlašuji, že jsem celou svou bakalářskou práci, včetně všech příloh, vypracoval samostatně.

4.5.2010

……

Chtěl bych tímto poděkovat svému vedoucímu, panu Ing. Vítězslavu Novákovi, Ph.D., za cenné rady a tipy, které jsem zúročil při psaní své práce.

Obsah

1 Úvod

V dnešním světě každá firma vyvíjející obchodní činnost přijímá či sama vystavuje faktury za poskytnuté služby, příp. odebrané zboží. To se dá realizovat v podstatě pouze třemi způsoby. Tím prvním je písemná forma, tj. klasický dopis zaslaný poštou, druhým je pak fax a tím třetím, v dnešní době velmi populárním a moderním způsobem, je posílání vystavených faktur zákazníkovi přes globální síť Internet, zejména s využitím elektronické pošty.

Vše má své pro a proti, ale peníze jsou pro každou společnost alfa omegou a vždy se proto snaží nacházet úsporná řešení. Poslat obyčejný dopis nestojí mnoho, ale pro velké firmy, které mají tisíce či miliony klientů, to může znamenat nemalé finanční náklady. Stejně tak i pro malé firmy jsou to zbytečné výdajové položky, a tak stále více roste obliba zasílání tohoto dokladu v elektronické podobě. Tento trend lze vysledovat např. u mobilních operátorů či bank.

Výhody jsou tedy hlavně ve snížení nákladů, časové náročnosti na odesílání, ale stejně tak i agendy, jelikož musel být někdo, kdo danou fakturu vytisknul, dal do obálky, nalepil známku a poslal. Samozřejmě u velkých podniků je to plně automatizovaný proces, ale pro střední a menší podniky to znamenalo čas a náklady. Mezi výhody lze ještě koneckonců zařadit i šetření přírodních zdrojů, což je další důvod, který naklání pomyslný jazýček vah na stranu využívání elektronického způsobu. Jistou nevýhodu elektronické pošty (dále jen „email“) oproti klasickému dopisu lze spatřovat v tom, že příjemce musí mít přístup na Internet a také mít nainstalovaný příslušný program. Ovšem v dnešní době je již pouze minimální procento firem bez připojení na Internet a bezplatného softwaru i pro komerční účely existuje řada. Navíc je zde vždy možnost využít zbylých dvou zmiňovaných způsobů.

Byl jsem v tomto směru osloven firmou Graphic4web, u které jsem vykonával svou školní praxi, jestli bych mohl vytvořit nějaké řešení, jež by nahradilo jejich

dosavadní, které nebylo zcela vyhovující. Jelikož již mají zavedený vzhled své faktury, který chtěli rozhodně zachovat, tak hlavní požadavek byl kladen na maximální podobnost výstupu.

Má bakalářská práce si tedy klade za cíl vytvořit uživatelsky přívětivý a efektivní program na tvorbu vystavovaných faktur, jehož výstupem bude soubor maximálně podobný tomu doposud používanému. Daný soubor bude pak zasílaný zákazníkům elektronickou poštou v takové podobě, aby nebyl editovatelný, ale zároveň aby z něj šel extrahovat text, typicky zkopírováním.

2 Popis současného stavu

Tato kapitola se bude nejprve věnovat samotné firmě Graphic4web a dále popíše dosavadní řešení včetně jeho nevýhod a důvodů hledání jiného řešení.

2.1 O firmě Graphic4web

Graphic4web je menší firma z Nového Jičína, působící v oblasti tvorby www stránek a webových prezentací. Je to relativně mladá firma pohybující se na trhu přes 6 let. Za tu dobu nasbírala bohaté zkušenosti a dokázala si získat pozici jedné z nejlepších firem v oboru v tomto regionu.

Její aktivity pokrývají oblasti od návrhu designu, zpracování validního a přístupného XHTML a CSS kódu, přes naprogramování funkčních celků, registraci domény, webhostingu, celkového zpřístupnění ze sítě Internet, zpracování dalších reklamních a propagačních materiálů, až po rozsáhlé firemní weby a portály nebo různé databázové či vyhledávací systémy s možností vlastní administrace. Kromě toho společnost poskytuje i další důležité služby spojené s vytvářením webových prezentací, jako je správné zvolení webhostingu a registrace domény, registrace stránek do katalogů a vyhledávačů a grafické zpracování bannerů. Dále se také zabývá tvorbou grafických návrhů vizitek, reklamních panelů, tabulí, letáků, firemních identit, razítek, potisků CD a dalších grafických služeb spojených s prezentací firmy nejen na Internetu.

Firma pro svou práci používá profesionální software Creative Suite CS4 od firmy Adobe Systems. Technologie, které používá, jsou v prvé řadě XHTML ve spojení s CSS a skriptovací jazyk PHP s využitím MySQL databází. Využívá také řešení, která jsou založena na technologiích jako jsou JavaScript, AJAX, Flash a další. Nedílnou součástí je i optimalizace pro vyhledávače pomocí SEO.

2.2 Dosavadní řešení

Zadavatel nejprve používal způsob zasílání faktur klasickým dopisem, avšak z hlediska finančního i časového (myšleno ve smyslu doby trvání doručení) se přeorientoval na vystavování faktur v elektronické podobě. K tomuto účelu dosud využíval kancelářský balík iWork[1], produkt firmy Apple[2]. Tento balík obsahuje desktopovou aplikaci Numbers (obdoba tabulkového procesoru Excel firmy Microsoft), ve které lze mj. vytvářet faktury – nabízí pro to i různé šablony. Následně zadavatel převáděl výstup z programu do PDF. Ovšem z hlediska uživatelského komfortu mu vadilo, že musel vyplňovat zbytečně mnoho položek a rovněž nebyl spokojen s absencí kontroly zadaných údajů. Např. se již párkrát stalo, že vinou ručního zadávání uvedl špatné datum splatnosti apod. Navíc po přidání další fakturované položky se mu nezkopírovaly funkce pro přepočet, takže je musel znovu nastavovat. Zkoušel proto i různé nekomerční produkty, ale žádný mu plně nevyhovoval.

Z jeho pohledu byla tato situace nežádoucí. Jeho přáním bylo mít jednoduchou a efektivní aplikaci, kde by zadal jenom to nejnutnější a ona by se o vše ostatní postarala. Proto hledal řešení, které by bylo pro něj „šité na míru“.

[1] Více na http://www.apple.com/iwork/.
[2] Zadavatel používá k práci produkty této firmy. Konkrétně notebook MacBook Pro s Mac OS X.

3 Teoretická východiska

Tato kapitola se zabývá obecně fakturou a ve zkratce všemi podstatnými technologiemi použitými při tvorbě programu.

3.1 Co je to faktura

Obecně řečeno, faktura je daňový doklad, který vystavuje jedna firma druhé kvůli tomu, aby dostala zaplaceno za provedené služby či odebrané zboží. [10] V dnešní digitální éře nemusí mít pokaždé hmotnou podobu, tedy onen vytištěný papír, mnoho firmem preferuje elektronickou podobu ve formátu PDF, příp. i obrázek ve formátu PNG či JPG. V takovém případě jsou, k vyhovění všech zákonných náležitostí, dvě následující možnosti:

- Jestliže je faktura zaslána emailem v PDF souboru neoznačeném digitálním podpisem (nutno podotknout, že elektronická faktura zaslaná emailem nemusí být vždy označena digitálním podpisem), nabízí se příjemci dvě možnosti, jak takovou fakturu zpracovat a uchovat. Tou první je, že si ji vytiskne, opatří jí podpisem osoby, která fakturu zaúčtovala a následně v papírové podobě uloží do archívu. Nebo, možnost druhá, ponechá si ji v elektronické podobě (PDF dokument nebo obrázek PNG, JPG) s tím, že jí sám opatří digitálním podpisem a uloží do svého elektronického archívu.

- V případě, že faktura zaslaná emailem v PDF souboru již je opatřena digitálním podpisem, má příjemce opět dvě možnosti. Buďto si fakturu vytisknout na papír, podepsat a archivovat v tištěné, hmotné podobě nebo si takto signovaný doklad rovnou uloží do svého elektronického archívu.

Je třeba mít na mysli, že elektronická data nejsou sama o sobě daňovým dokladem. Tím kýženým dokladem se stávájí až tehdy, kdy je zajištěn jejich původ neboli *autenticita* a jejich nezměnitelnost neboli *integrita*.

3.1.1 Formální náležitosti faktury

V každé vystavené faktuře musí být uvedeny následující údaje:

- označení „Faktura“ a pořadové číslo,
- jméno a adresa vystavovatele,
- jméno (příp. název firmy) a adresa příjemce,
- datum vydání a datum splatnosti,
- v případě, že je faktura určena plátci DPH, musí obsahovat i datum zdanitelného plnění,
- podpis vystavovatele faktury a razítko není nutné uvádět.

3.1.2 Proforma faktura

Jedná se o obecně používaný název, pod kterým se označuje neúčetní písemnost, která se vystavuje ještě před skutečným daňovým dokladem, jako je třeba faktura. I pro plátce DPH je tato písemnost neúčetním dokladem.

Proforma faktura obsahuje zpravidla všechny náležitosti jako skutečný daňový doklad. Údaje v ní uvedené jsou však pouze informativního charakteru a k tomuto účelu byla de facto i vytvořena – informovat, resp. sloužit jako podklad pro rozhodnutí.

3.2 HTML

„*HTML je značkovací jazyk pro hypertext. Je jedním z jazyků pro vytváření stránek v systému World Wide Web, který umožňuje publikaci dokumentů na Internetu*.“ [12] Za jeho vznikem stojí Sir Tim Berners-Lee, který také navrhl protokol HTTP, napsal první webový prohlížeč WorldWideWeb a koncem roku 1990 spustil první webový server na světě – http://info.cern.ch/. Poté v říjnu roku 1994 založil

nezávislé mezinárodní konsorcium W3C, jehož je dodnes ředitelem, které dohlíží na další vývoj Webu.

Pro svou jednoduchou syntaxi a efektivnost si získal velmi rychle dominantní pozici ve světě. Pro demonstraci – základní struktura každé HTML stránky:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Final//EN">
<HTML>
  <HEAD>
    <TITLE> </TITLE>
    <META>
  </HEAD>
  <BODY>
        <H1> </H1>
  </BODY>
</HTML>
```

K pochopení principu tohoto jazyka stačí pochopit dvě jednoduchá pravidla. První se týká *tagů* (značek) a druhé *odkazů*.

3.2.1 Tagy

Všechny příkazy jakyka HTML jsou tagy (je možno užívat také termínu „elementy“) – vše, co je mimo ně, je vlastní obsah stránky. Tagy se zapisují do ostrých závorek a mohou také obsahovat různé atributy. Pomocí příslušných tagů by se měly označovat jednotlivé části textu podle toho, co představují (nadpisy, odstavce, seznamy apod.). Rozlišují se dva druhy značek – *párové* a *nepárové*. Tím častějším jsou párové tagy, to znamená, že mají jednu část pro začátek a jednu pro konec. Lépe to osvětlí ukázka:

```
<H1> Toto je párový element označující nadpis první úrovně. </H1>
```

Nepárové značky jsou již ze své podstaty nedělitelné a tudíž do nich nelze vložit text. Jedny z nejpoužívanějších elementů jsou zařádkování a tlačítko:

```
<BR> Tímto se zářádkovává text. A takto vypadá potvrzovací tlačítko:
<INPUT TYPE="submit" VALUE="Potvrdit">
```

3.2.2 Odkazy

Druhé pravidlo se týká jedinečného principu odkazů (angl. „link" nebo „hypertext link"). Odkazy, jak již název sám napovídá, slouží k odkazování na jiná data, která se nemusí nutně nalézat v aktuálním dokumentu. Typickým příkladem je odkazování se na jinou stránku nebo také „sem vlož obrázek, který se nachází tam a tam", místo aby vsadil do textu binární data, tedy obrázek samotný. Názorná ukázka:

```
<a href="http://www.example.com"> Přejdi na stránku Google </a>
```

Obecně k jazyku HTML je třeba ještě dodat, že nedostačoval moderním požadavkům a tak jeho poslední verzí 4.01 byl vývoj ukončen. Vládu po něm převzal jeho nástupce XHTML[3].

3.3 XHTML

XHTML (v překladu „rozšiřitelný hypertextový značkovací jazyk") [18] je poměrně nový značkovací jazyk vycházející z XML[4] a HTML (odtud taky plyne jeho název). Je založen na pevném dodržování standardní syntaxe a jsou v něm stanovena daleko pevnější pravidla než v klasickém HTML. Hlavní rozdíly XHTML oproti HTML tedy spočívají v tom, že v XHTML musí být všechny elementy ukončené a to včetně nepárových, jako jsou `<meta>`, `<br>`, `<hr>`, `<img>` ad. Zápis je možno

[3] V současné době již existuje „pracovní návrh" (angl. „working draft") HTML 5, který s největší pravděpodobností převezme zpět vládu po XHTML.
[4] XML (česky „rozšiřitelný značkovací jazyk") je značkovací jazyk, ovšem jeho cílem je pouze definovat logickou strukturu dat, nikoliv jejich formátování.

provést více způsoby. Buď klasicky <meta></meta> nebo zkráceně <meta/> (používanější způsob). Mezi další rozdíly patří povinnost psát všechny značky a jejich atributy malými písmeny, jelikož XHTML je case sensitive (v překladu tzn., že záleží na velikosti písmen). Do výčtu ještě zbývá zmínit, že všechny hodnoty atributů musí být uzavřeny do uvozovek a každý dokument musí začínat XML deklarací.

Základní struktura každé XHTML stránky: [19]

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="cs" lang="cs">
    <head>
        <meta http-equiv="content-type" content="text/html;
          charset=UTF-8" />
       <meta http-equiv="content-language" content="cs" />
       <title> </title>
    </head>
    <body>
        <h1> </h1>
    </body>
</html>
```

Ve srovnání se stejnou stránkou napsanou ve starším jazyce HTML, je tato rozšířena o několik povinných atributů ve značce <html>, delším <!DOCTYPE> a dvěma <meta/> tagy.

Cílem tohoto jazyka je zobrazení dokumentu nezávisle na platformě. V praxi to například znamená bezproblémové prohlížení webových stránek v prohlížeči mobilního telefonu, relevantnější výledky vyhledávání ve vyhledávačích či zrychlení přenosu dat.

3.4 CSS

Každá stránka má svůj obsah a formu. A platí, že co nepatří do obsahu, je forma. Pod pojmem forma je v tomto smyslu myšleno formátování, tj. barva písma, pozadí, velikost nadpisů apod. Během vývoje jazyka HTML vznikly dva různé způsoby, jak formátovat text:

- Původní myšlenka HTML počítala s tím, že text bude vždy formátován přímo na místě pomocí značek, např. `<h1>` pro nadpis, `<b>` pro tučné písmo, `<center>` pro vycentrování apod. Ovšem, jak se později ukázalo, toto řešení má svá omezení.
- Lepší a modernější způsob je použití tzv. *kaskádových stylů* (CSS), jejichž výhody budou popsány níže.

CSS spatřilo světlo světa někdy okolo roku 1997 a od té doby se stává nedílnou součástí tvorby www stránek. V současné době je ve verzi 2.1, ale již zanedlouho bude uvedena nová, v mnohých ohledech revoluční, verze 3.0. Základní koncept je velmi jednoduchý – možnost modifikovat výchozí vlastnosti stylů, které již v HTML existují, a také vytvářet nové styly, kterým lze přiřazovat určité vlastnosti. Dalším klíčovým aspektem je v názvu slovo „kaskádové“. To je tam proto, protože se na sebe mohou vrstvit definice stylů, ale vždy platí jenom ta poslední. [2]

Existují tři různé deklarace stylů:

- Přímo v elementu pomocí atributu style="…". Např. když má mít odstavec modrý text o velikosti 16 pixelů s obrázkem na pozadí, tak zápis by vypadal následovně:

```
<p style="color: blue; font-size: 16px; background-image: url( 'obr.jpg' )"> Jsem
    modrý text o velikosti 16px s obrázkem na pozadí. </p>
```

I když je tento způsob zcela v pořádku, tak např. při delším textu s mnoha odstavci, které mají mít stejné vlastnosti, je velmi nepraktické a neefektivní psát to pořád dokola. Tudíž tento způsob se doporučuje nepoužívat.

- Pomocí „stylopisu“ (angl. „stylesheet“), který se vkládá přímo do hlavičky stránky a píše se mezi značky `<style>` a `</style>`. Je v něm obecně napsáno, co má být jak zformátováno. Opět názorná ukázka:

```
<head>
    <style type="text/css">
        p { color: blue; font-size: 16px; background-image: url( 'obr.jpg' ) }
    </style>
</head>
<body>
    <p> Jsem modrý text o velikosti 16px s obrázkem na pozadí. </p>
    <p> Já jsem taky modrý text o velikosti 16px s obrázkem na pozadí. </p>
</body>
```

- Třetím a nejpoužívanějším řešením je použití externího stylopisu. Je to soubor s příponou *.css* (typicky *styly.css*), ve kterém je umístěn stylopis a na který se stránka odkazuje značkou `<link>`. Hlavní výhoda spočívá v tom, že tento soubor lze přiřadit libovolnému počtu stránek, takže pak všechny vypadají podobně. Příklad, jak vypadá nalinkování externího souboru pojmenovaného *styly.css*:

```
<head>
    <link href="styly.css" rel="stylesheet" type="text/css" />
</head>
<body>
    <p> Jsem modrý text o velikosti 16px s obrázkem na pozadí. </p>
    <p> Já jsem taky modrý text o velikosti 16px s obrázkem na pozadí. </p>
</body>
```

V externím souboru pak bude:

```
p { color: blue; font-size: 16px; background-image: url( 'obr.jpg' ) }
```

3.5 Java

Java (foneticky „džava“) je objektově orientovaný jazyk, vyvinutý a v současnosti stále vyvíjený společností SUN Microsystems. Základní myšlenkou bylo vytvořit systém pro tzv. „embedded“[5] zařízení. Tým inženýrů z firmy SUN původně zvažoval použití jazyka C++, ale z různých důvodů byli nuceni tuto cestu zavrhnout (např. kvůli chybějícímu garbage collectoru, vláknům, serializaci objektů ad.). Nakonec také chtěli platformu, která by byla lehce přenositelná na všechny druhy zařízení.

Proto začali okolo roku 1991 pracovat na svém projektu, pojmenovaném „The Stealth Project“ (v překladu „Tajný projekt“), který byl ale záhy přejmenován na „Green Project“. Vytvořili zcela nový jazyk, kterému dali jméno Oak (česky „dub“) – údajně proto, že jim tento strom rostl pod okny kanceláře. V roce 1995 ovšem zjistili, že jazyk s názvem Oak již existuje a tak bylo nutné dát jazyku jiný název. Inspirací pro nové jméno se pak stala cesta jednoho z tvůrců do bufetu na kávu. Java v americkém slangu je označení pro kávu.

První verze byla oficiálně představena v květnu roku 1995 a v roce následujícím byl vydán i první JDK 1.0 (Java Development Kit). Java se postupně vyvíjela a stala se jedním z nejpoužívanějších programovacích jazyků na světě. V roce 2007 společnost SUN uvolnila zdrojové kódy Javy (čítající cca 2,5 miliónů řádků kódu) a od toho okamžiku je dále vyvíjena jako open-source.

Java je jazykem interpretovaným, z čehož plyne, že místo skutečného strojového kódu se při kompilaci vytváří pouze bajtový kód (ByteCode), který je spouštěn na

[5] Slovem „embedded“ se obecně označuje počítačový systém nebo elektronické zařízení, které provádí vyhrazenou funkci, např. domácí spotřebiče.

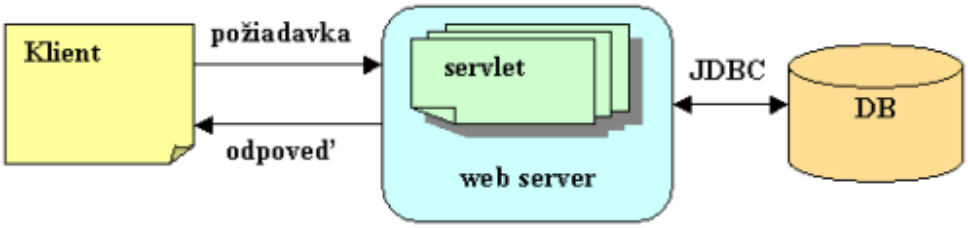
JVM (virtuálním stroji Javy). Ke spuštění Java aplikace na počítači je zapotřebí si nainstalovat JDK (ze stránek http://java.sun.com), který v sobě obsahuje JRE[6].

Hlavní výhodou Javy je její přenositelnost a platformní nezávislost. V praxi to znamená, že může pracovat na rozličných systémech počínaje čipovými kartami (platforma JavaCard), přes mobilní telefony (platforma J2ME), desktopové aplikace (platforma J2SE) až po robustní distribuované systémy (platforma J2EE). Tyto technologie se jako celek nazývají „Java platforma“. Mezi Java technologie patří také mj. *Java Servlety*, *Java Server Pages*, *Expression Language* a *JSTL značky*.

3.5.1 Java Servlety

Jedna z definic o servletech říká, že *„se jedná o program, který běží na webovém serveru a působí jako střední vrstva mezi požadavkem přicházejícím z webového prohlížeče nebo od jiného HTTP klienta a databází či aplikací na serveru HTTP.“* [3] Viz obr. č. 3.1.

Obr. 3.1 Zjednodušená komunikace servletu s klientem a databází



Zdroj: Procesy mezi klientem a serverem [online]. [cit. 9.4.2010]. Dostupné na WWW: <http://interval.cz/clanky/java-servlets-predstavenie-technologie>.

Zjednodušeně se dá říci, že servlet je programová komponenta napsaná v jazyce Java, která běží na webovém serveru a jejíž hlavními úkoly jsou zpracovávat HTTP požadavky a generovat HTTP odpovědi (typicky HTML stránky). Z technického hlediska se jedná o obyčejnou třídu, potomka abstraktní třídy *javax.servlet.http.HttpServlet*, řídící tok aplikace. Zatím poslední verze je 2.5.

[6] Java Runtime Environment poskytuje knihovny, Java Virtual Machine a další komponenty potřebné ke spuštění appletů a aplikací napsaných v Javě.

3.5.2 JSP stránky

Jedná se o technologii pro tvorbu dynamických webových stránek, která na první pohled v mnohém připomíná známější jazyk PHP. Jsou to tedy klasické HTML stránky, do kterých se vkládají speciální skriptovací značky obsahující javovský kód. Tyto značky jsou uvozovány znaky **<%** a **%>**. Princip činnosti spočívá v tom, že „*zdrojový kód stránky JSP je prekladaný na servlet a následne kompilovaný. Výsledkom je servlet, ktorý generuje HTML. To je poslané klientovi, ako odpoveď na jeho požiadavku*“. [13] Měnit dynamicky obsah stránek lze také použitím *Expression Language* či *JSTL značkami*.

3.5.3 Expression Language (EL)

„*Expression Language is a scripting language which allows access to Java components (JavaBeans) through JSP*.“ [9] Jazyk EL umožňuje snadný přístup k datům, uloženým jako atributy v rozsahu stránky (page), požadavku (request), sezení (session) nebo aplikace (application).

Používá se pro: [5]

- dynamické čtení dat uložených v komponentách JavaBeans, polích a kolekcích a implicitních objektech,
- dynamický zápis dat do komponent JavaBeans,
- volání metod,
- dynamické vykonávání aritmetických operací.

Výrazy jazyka EL se vkládají mezi složené závorky uvozené znakem pro dolar – **${ }**. Jsou dva možné způsoby zápisu:

```
<p> ${ objekt.vlastnost } </p>
<p> ${ objekt[ ‘vlastnost‘ ] } </p>
```

V obou případech platí, že pro první část výrazu se hledá proměnná v rozsahu page, request, session, příp. application s klíčem „objekt“ a druhá část výrazu znamená zpřístupnění části daného objektu.

3.5.4 Standardní JSTL značky

JSTL je knihovna standardních značek, které řeší nejčastěji používané akce při vývoji JSP. Vychází z jazyka EL a rozděluje se na několik základních skupin:

- *core* (s prefixem „c“) – do této skupiny patří tagy pro vykonávání podmínek, cyklů, správu proměnných v různých rozsazích, práci s URL a odchytávání výjimek,
- *formatting* (s prefixem „fmt“) – tato skupina zajišťuje formátování čísel, zpráv a datumů,
- *XML* (s prefixem „x“) – zpřístupnění struktury XML, transformace pomocí XSLT,
- *funkce* (s prefixem „fn“) – manipulace s řetězci a délka kolekce,
- *SQL* (s prefixem „sql“) – umožňuje vykonávat dotazy nad relačními databázemi a zpřístupňuje jejich výsledky.

Pro využívání těchto tagů je zapotřebí si na JSP stránce importovat knihovnu JSTL pomocí direktivy „taglib“ (Tag Library Descritor) s příslušným prefixem. Typický zápis vypadá následovně:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

3.6 JavaScript

JavaScript se díky svému názvu často plete s Javou. Ačkoliv oba mají základní konstrukce stejné, jedná se o dva rozličné jazyky. JavaScript je běžně používán

jako skriptovací jazyk, tudíž se zpracovává na straně klienta (na rozdíl od Javy). V praxi to znamená, že server posílá klientovi HTML stránku se skriptem a ten se teprve tam vykoná – za předpokladu, že klient má v prohlížeči povolen JavaScript. Dále by se dal charakterizovat jako jazyk interpretovaný (nemusí se kompilovat), case sensitive (rozlišuje malá a velká písmena) a dokonce je i objektový. Nevýhodou pak může být nemožnost ukládat data a přistupovat k souborům (dá se to obejít pomocí cookies[7]). To vše ho činí použitelným de facto jen na webových stránkách.

3.7 IDE NetBeans

Pod zkratkou IDE se v českém překladu skrývá pojem „integrované vývojové prostředí“. Jde o vysoce propracovaný a sofistikovaný software usnadňující práci programátorům, zpravidla zaměřený na jeden programovací jazyk. Každé takové vývojové prostředí obsahuje editor zdrojového kódu, kompilátor a většinou také debugger. [17]

Vlajkovou lodí firmy SUN, duchovního otce Javy, je IDE NetBeans. Není bez zajímavosti, že za vznikem tohoto open-source projektu v roce 1996 stáli čeští studenti z UK[8]. Šířen je pod licencí CDDL a GNU General Public License – jinými slovy, je možné jej bezplatně používat pro komerční i nekomerční účely. [8]

3.8 Webový kontejner Apache Tomcat

„*Apache Tomcat is an open-source servlet container developed by the Apache Software Foundation*“. [7] Je to jeden z nejznámějších a nejpoužívanějších kontejnerů implementující v sobě technologie *Java Servletů* a *JSP*. Je oblíbený pro svou jednoduchost, transparentnost a nenáročnost na systémové prostředky.

[7] Cookies jsou malé textové soubory, které si server ukládá na straně klienta k jeho budoucí identifikaci.
[8] Univerzita Karlova v Praze

3.9 Adobe Acrobat

Adobe Acrobat je produkt od firmy Adobe Systems pro tvorbu profesionálních dokumentů ve formátu PDF, které je možno vytvářet, prohlížet, upravovat a případně i jinak zpracovávat. Poslední verze umožňuje také např. vytvářet dynamické formuláře. Kromě výše zmíněného, tento program umí také vytvořit ovladač pro virtuální tiskárnu, díky které lze tisknout dokumenty do formátu PDF. K vytvoření takového PDF souboru stačí v jakémkoliv programu, který podporuje tisk, vytvořit požadovaný dokument a namísto normální tiskárny zvolit tu virtuální (v tomto případě Adobe PDF) a dokument vytisknout. [6]

3.10 Microsoft Visio

„Usnadňuje odborníkům z oblasti IT a dalších odvětví vizualizovat, analyzovat a komunikovat komplexní informace, systémy a procesy." [15] Jinými slovy jde o nástroj na kreslení schémat a v této práci bude použit na tvorbu UML[9] diagramu.

[9] UML (česky „unifikovaný modelovací jazyk") diagramu se využívá zejména pro návrh objektově orientovaných programů, příp. pro podrobnou specifikaci procesní analýzy.

4 Návrh a realizace programu

V této kapitole se budu nejprve věnovat hledání různých možností realizace zadání a to tak, aby splňovaly požadavky zadavatele. Poté se zaměřím na samotnou realizaci projektu.

4.1 Seznam požadavků zadavatele

Ze strany zadavatele byly uvedeny následující požadavky:

- výsledný PDF soubor by měl vypadat maximálně stejně jako ten dosavadní, viz příloha 1 Původní faktura,
- číslo faktury bude desetimístné. Při otevření programu se samo nastaví číslo faktury, resp. 5. – 10. znak, což je aktuální datum ve formátu ddMMyy. První 4 znaky je pořadové číslo, které se nastavuje ručně,
- číslo faktury se automaticky uvede i v červeném řádku jako variabilní symbol, viz příloha 1 Původní faktura,
- na základě aktuálního data se automaticky nastaví datum vystavení, datum uskutečnění zdanitelného plnění a splatnost (ta se počítá jako aktuální datum + 7 dní),
- způsob dopravy bude jako select box pro výběr z možností „Zasláno na e-mail odběratele“ a „Nakopírováno na server“,
- údaje v sekci „Odběratel“ se budou automaticky kopírovat do sekce vedle, kde je doručovací adresa,
- možnost vložit až 5 položek k fakturaci s tím, že se bude zadávat jen cena bez DPH, ostatní výpočty se budou počítat automaticky,
- uvést i položku „Zaplacená záloha“, kde se bude zadávat cena včetně DPH, která již byla zaplacena a tato záloha se bude odečítat z celkové hodnoty faktury,

- doplnit možnost „Sleva“, kam bude možno zadat procentní slevu, která se projeví na cenách bez DPH a celkových propočtech,
- tabulka s rekapitulací cen umístěná v dolní části bude automaticky přepočítávána dle položek na faktuře.

Zjednodušeně by šlo říci, že z výše uvedených požadavků je hlavní důraz kladen primárně na výstup, tedy aby výsledný soubor v PDF byl maximálně podobný tomu dosavadnímu. Z dalších požadavků pak vyplývá, že uživatel má zadat pouze základní vstupní údaje a zbytek, tj. hlavně propočty, má za úkol zpracovat samotný program. To je samozřejmě správná cesta, jelikož za tímto účelem byly přístroje, potažmo programy, vymyšleny a vyrobeny – provádět mechanické propočty. Tím se také sníží možnost chyby způsobené lidským faktorem. Sekundárním požadavkem je pak efektivita práce s programem, pod čímž se myslí uživatelská přívětivost programu, stabilita a nenáročnost na zdroje. Způsob, jakým by aplikace měla být zrealizována, byl ponechán čistě na mne. Měl jsem tedy volnou ruku v tom, jakou formu a technologie zvolím. Zároveň jsem nemusel řešit archivaci vytvořených faktur, jelikož pro to už mají ve firmě svůj zavedený způsob. Odpadá tím nutnost vytvořit databázi a ukládat do ní data.

4.2 Analýza a volba způsobu implementace

Cílem mého snažení je tedy vytvořit program na tvorbu faktur dle požadavků zadavatele. Při hledání způsobu jakým směrem se vydat, či jakou technologii zvolit, bylo třeba mít stále na paměti, že zadavatel používá operační systém od firmy Apple (Mac OS X). Tím pádem bylo nutné zvolit takové řešení, které by bylo přímo dělané pro daný OS nebo najít nějaké *univerzální, platformě nezávislé řešení*. Rozhodl jsem se pro druhou možnost, a to ze dvou důvodů. První vychází z toho, že univerzální řešení je ad hoc lepší a ten druhý důvod je čistě praktický – nevlastním žádný produkt firmy Apple a tudíž by realizace projektu byla poněkud problematická.

První myšlenka padla na realizaci celého projektu v Javě, konkrétně v J2SE s použitím komponentů knihovny Swing. Po zralé úvaze jsem byl nucen ji opustit, jelikož by bylo poměrně náročné vytvořit takový grafický výstup, jaký byl striktně požadován a rovněž i následný převod do PDF by byl komplikovaný. Nabízela se další možnost jak realizovat projekt v Javě, a sice použitím open-source knihovny JasperReports[10]. Vzhledem k náročnosti práce s touto knihovnou jsem se rozhodl hledat jinou cestu. Jako třetí možnost se nabízelo vytvořit webovou aplikaci, tedy webové stránky, což by výše zmíněným požadavkům vyhovovalo ve všech směrech – univerzální a efektivní řešení, které lze snadno vyexportovat do PDF.

Nakonec tedy zvítězila třetí možnost – realizace projektu pod tenkým klientem. Jinými slovy, aplikace běží na serveru, který komunikuje s uživatelem skrz webový prohlížeč. Celý projekt je založen na kombinaci technologií *Javy* (ta se postarala o aplikační logiku), *XHTML* s *CSS* (pro grafický výstup) a softwaru *Adobe Professional* (pro převod do PDF). Vývoj celé aplikace jsem realizoval ve vývojovém prostředí NetBeans.

4.2.1 Výběr serveru

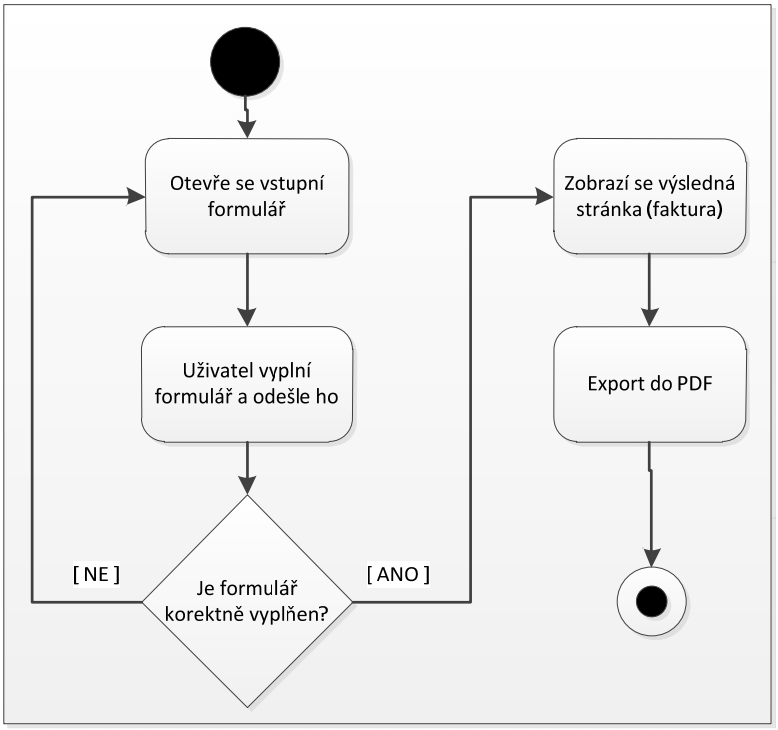
Výsledná webová aplikace, tedy www stránky, se obyčejně umisťuje na Internet, aby byla přístupná všem uživatelům, kteří mají o dané stránky zájem. To zároveň znamená vybrat vhodný webhosting, který by zajišťoval provoz webových stránek. Ovšem v tomto případě je situace opačná – zadavatel si nepřeje vystavovat výslednou aplikaci na Internet, jelikož pro to není žádný důvod a také z hlediska bezpečnosti (aby někdo nepovolaný nemohl neoprávněně vystavovat faktury jménem firmy) je to racionální rozhodnutí. Aplikace bude tedy provozována pouze na lokálním serveru, čímž také odpadá obvyklý problém výběru webhostingu i případných nákladů spojených s touto službou. Celá aplikace bude provozována ve webovém kontejneru Apache Tomcat (dále jen „Tomcat“), který se perfektně hodí pro potřeby tohoto projektu – je rychlý, multiplatformní a zdarma.

[10] JasperReports je celosvětově nejpopulárnější reportovací nástroj kompletně napsaný v Javě. Domovská stránka http://jasperforge.org/projects/jasperreports.

4.3 Návrh programu

Důležitým prvkem každého programu je být „user-friendly“ (česky „uživatelsky přátelský, příjemný“). To v zásadě znamená, že nehledě na to, co daný program umí, by měl být natolik jednoduchý a přehledný, aby se v něm uživatel bez problému hned zorientoval a mohl ho začít používat. S tím souvisí i forma provedení (vzhled) – množství grafických prvků, barvy, použitý font písma apod. S ohledem na výše uvedené se mi jevilo jako nejlogičtější a nejefektivnější udělat nejprve vstupní **formulář**, kde uživatel zadá potřebné údaje, ten se pak odešle na **servlet**, který zkontroluje, zdali jsou všechny položky korektně vyplněny. V případě, že jsou v požadovaném tvaru, servlet přesměruje požadavek na další stránku, kde již bude výsledná **faktura**, samozřejmě zatím ve formátu HTML stránky. Poté bude stačit pouze vytisknout tuto stránku na virtuální tiskárně, čímž se stane faktura kýženým **PDF dokumentem** připraveným k archivaci a odeslání. Lépe to osvětlí obrázek 4.1 Diagram aktivit.

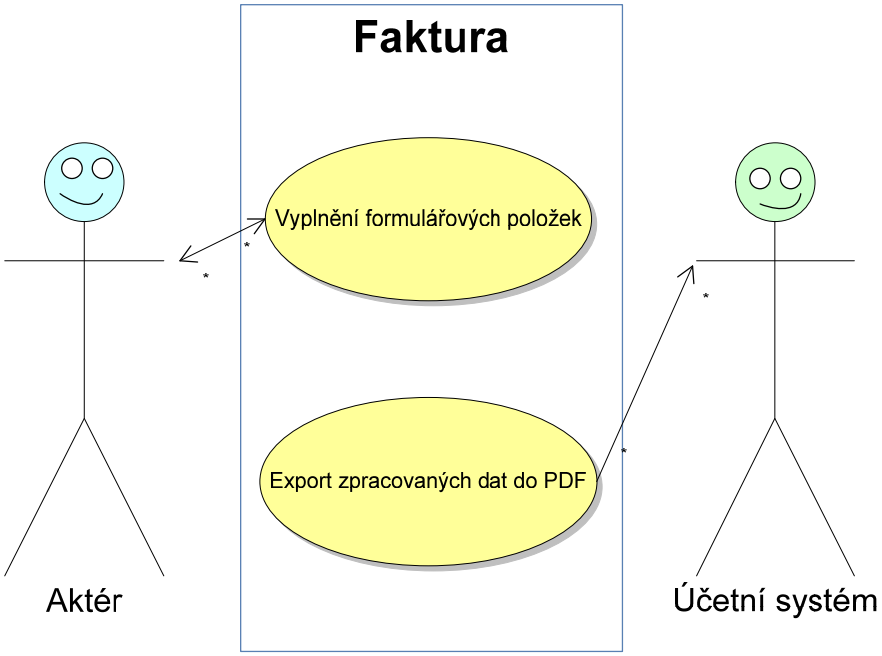
Obr. 4.1 Diagram aktivit



4.3.1 Use Case model v UML

Model Use Case (česky „případ užití“) představuje popis funkcí aplikace a její použití z hlediska uživatelských rolí, viz obr. 4.2. Aktér (vystupuje v systému jako primární aktér) zadává vstupní údaje a tím spouští případ užití. Účetní systém nevstupuje do systému (je tedy uživatelem), avšak je na něm zainteresován – faktury se musí archivovat.

Obr. 4.2 Use Case model aplikace



4.3.2 Program z pohledu uživatele

Každá aplikace je vytvářena pro uživatele, protože on je ten, kdo s ní bude pracovat. Konkrétně v tomto případě je z jeho pohledu nejdůležitější, aby taková aplikace jednak vypadala hezky a jednak aby se nemusel starat, jestli vyplnil všechny položky korektně. Proto se při spuštění aplikace zobrazí jednoduchý formulář, viz obr. 4.3 níže, po jehož vyplnění chce uživatel vidět výslednou fakturu, kterou už jenom převede do PDF.

4.3.3 Program z pohledu programátora

Na rozdíl od uživatele musí programátor nahlížet na program z diametrálně odlišného pohledu. Uživatel neřeší tzv. „what if“ neboli „co se stane, když“, to má na starosti právě programátor. Ten musí zajistit a ošetřit všechny možné stavy, které mohou nastat – typicky když uživatel zadá např. alfabetický znak do textového pole, které je určeno pouze pro numerické znaky, jak ilustruje obrázek v příloze č. 2.

Blíže o této problematice a celém postupu tvorby aplikace bude pojednáno v následujících podkapitolách.

4.4 Realizace programu

Na začátku realizace jsem si vytvořil projekt s názvem Faktura, který implicitně vytvoří stránku index.jsp, jejíž obsah je prakticky shodný s klasickou HTML stránkou. Ovšem o aplikační logiku se měl starat servlet, tudíž účelem této stránky bylo pouze přesměrovat právě na něj. To jsem provedl pomocí JSTL značek (k jejich používání bylo zapotřebí si importovat do projektu knihovnu JSTL 1.1), kdy jsem do těla stránky vložil následující kód:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:redirect url="Servlet" />
```

Poté jsem si vytvořil třídu **Servlet** (zabalenou v balíku „tridy“), která, jak již bylo výše zmíněno, má na starosti aplikační logiku. V tomto případě to znamená, že přijímá vstupní data z formuláře, která následně zpracuje a na základě toho vyhodnotí, kam přepošle požadavek neboli na jakou stránku přejde.

V servletu byly použity čtyři stěžejní metody:

- *processRequest(HttpServletRequest request, HttpServletResponse response)* – volá se při každém spuštění servletu. Obsahuje parametry

request a response (požadavek a odpověď) třídy HttpServletRequest, resp. HttpServletResponse. Vyhazuje výjimky ServletException a IOException.

- *getParameter(String param_name)* – slouží k získání dat z formuláře s názvem určitého parametru. V případě, že parametr neexistuje, metoda vrátí „null“, jinak vrací hodnotu typu String.

- *setAttribute(String name, Object object)* – používá se pro uložení objektu jako atribut do rozsahu request. První parametr je identifikátor, pod kterým bude možné v JSP stránce získat jeho hodnotu.

- *getRequestDispatcher("URL")* – tato metoda přesměrovává na relativní URL adresu (parametr musí být uvozen znakem pro lomítko „/“). Poté se na ni ještě musí zavolat metoda *forward(request, response)*, která předává objekty externímu zdroji, např. JSP stránce.

Všechny zmíněné metody jsem v aplikaci použil následujícím způsobem:

```
protected void processRequest(HttpServletRequest request, HttpServletResponse
        response) throws ServletException, IOException {
    String jmeno = request.getParameter( "jmeno" );
    request.setAttribute( "jmeno" , jmeno );
    request.getRequestDispatcher( "/WEB-INF/pages/form.jsp" ).forward(request,
    response); }
```

Při prvním zavolání servletu se uloží v metodě *processRequest()* zejména datum ve formátu do rozsahu request (pod názvem „date“) a přesměruje se na úvodní stránku programu – tedy zobrazí se formulář. V praxi to vypadá následovně:

```
request.setAttribute( "date" , new SimpleDateFormat( "dd.MM.yyyy" ).format( new
    GregorianCalendar().getTime() ) );
request.getRequestDispatcher( "/WEB-INF/pages/form.jsp" ).forward(request, response);
```

Zároveň se ještě uloží atributy s výchozími hodnotami pro zálohu, slevu, výši DPH a stát.

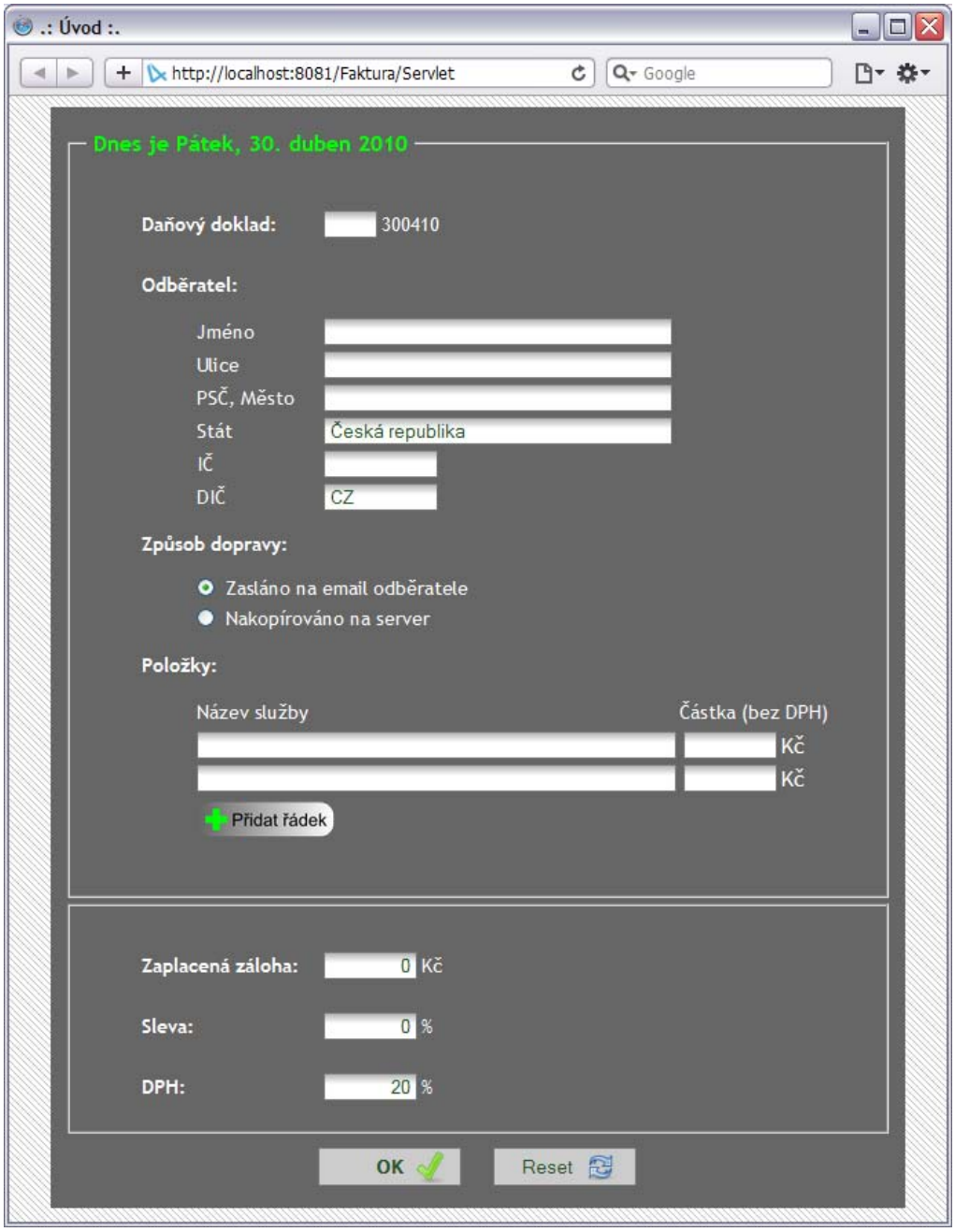
4.4.1 Popis formuláře a jeho funkcí

Pro vstupní formulář jsem si vytvořil stránku **form.jsp**, kterou jsem si rozvrhnul do dvou skupin, jak lze vysledovat z obr. 4.3. V první skupině jsou položky, které musí uživatel vždy vyplnit – číslo daňového dokladu, informace o klientovi, název služby a její cenu. Ve druhé skupině pak jsou položky, které se většinou nemění, resp. není třeba je vyplňovat – zaplacená záloha, sleva, popř. výše DPH. Pro optické rozdělení těchto skupin jsem použil element `<fieldset>`, který udělá vnější rámeček dané skupiny. Zároveň má tu výhodu, že se do něj dá vložit element `<legend>`, který slouží jako jakýsi popisek. V programu jsem do tohoto elementu vložil pomocí formátovacích značek knihovny JSTL aktuální datum, jež jsem si v servletu „připravil“ a nastavil do rozsahu request jako atribut „date“. Tento atribut jsem si na stránce vyžádal pomocí EL a naformátoval podle definovaného vzoru. Výsledek následující ukázky lze spatřit na obr. 4.3 v levém horním rohu formuláře:

```
<fieldset>
    <legend>
        <fmt:parseDate value="${ date }" pattern="d.M.yyyy" var="datum" />
        Dnes je <fmt:formatDate value="${ datum }" pattern="EEEE, d. MMMM yyyy" />
    </legend>
</fieldset>
```

Ve formuláři jsem se snažil přehledně seskupit všechny nejnutnější položky, které musí uživatel vyplnit, tj. *daňový doklad*, sekce *odběratel*, *způsob dopravy* a názvy *fakturovaných položek* s příslušnými *cenami*. Ostatní položky jako *zaplacená záloha*, *sleva* a *DPH*, jsou již implicitně nastaveny. Zadavatel se ve svých požadavcích nezmiňoval o nutnosti uvést i možnost DPH, ale vzhledem k tomu, že mohou kdykoliv v tomto směru nastat legislativní změny, navrhnul jsem ji zařadit, s čímž zadavatel souhlasil.

Obrázek 4.3 Vstupní formulář



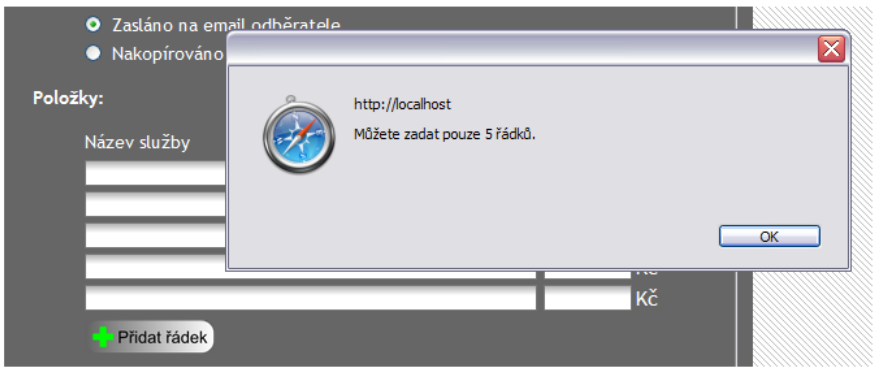
Pro výběr způsobu dopravy jsem si nejprve do stránky vložil skriplet, tj. Java kód, kde jsem si deklaroval proměnnou „vyber“, do které jsem následně vložil metodou *getParameter()* hodnotu atributu „vyber“. Při prvním spuštění stránky se tato proměnná vždy inicializuje na hodnotu „1“. Je to kvůli tomu, že „Způsob dopravy“ je

tvořen elementy `<input />` typu „radio“, které lze pomocí atributu „checked“ vybrat (označit). K určení, který element bude vybrán, jsem použil „výrazu“ neboli skriptovacích značek **<%=** a **%>**, kde ternární operátor na základě hodnoty proměnné „vyber“ vloží či naopak nevloží atribut „checked“. Tím se zaručí, že vždy bude vybrána jedna z možností. Zjednodušená ukázka to jistě lépe osvětlí:

```
<%  String vyber = request.getParameter( "vyber" );
            if ( vyber == null )
                vyber = "1";    %>
…
<input  type="radio"  name="vyber"  value="1" <%= ( vyber.equals("1") ?
    "checked=\"checked\"" : "" ) %> />
<input  type="radio"  name="vyber"  value="2" <%= ( vyber.equals("2") ?
    "checked=\"checked\"" : "" ) %> />
```

V požadavcích bylo rovněž uvedeno, že má být možnost vložit až pět položek k fakturaci. Většinou ovšem zadavatel fakturuje jednu až dvě, tudíž jsem navrhl uvést jako výchozí pouze dvě položky s tím, že bude možnost si přidávat potřebné řádky tlačítkem (resp. obrázkem) „Přidat řádek“. Toho jsem docílil použitím JavaScriptu, kdy po kliknutí na toto tlačítko se zavolá funkce *pridat_radek()*, která do formuláře dynamicky vloží požadovaný řádek. Při pokusu o přidání čtvrtého řádku (tj. celkově šestého) vyskočí hláška upozorňující na maximální počet řádků a další již nebude možno přidávat, viz obr. 4.4.

Obr. 4.3 Hláška při zadání více jak 5 řádků



Daňové identifikační číslo (dále jen „DIČ“) se v případě právnické osoby (např. společnost s ručením omezeným – dále „s.r.o.“) odvíjí od identifikačního čísla (dále jen „IČ“) – pouze s tím rozdílem, že DIČ začíná kódem země. Když potom bude klient zadavatele tuzemská firma s ručením omezeným (což je nejčastější případ) a bude mít IČ např. „123 45 678“, tak DIČ bude ve formátu „CZ12345678“. Proto jsem pro větší uživatelský komfort vložil do stránky JavaScriptovou funkci nazvanou intuitivně *zapis_do_DIC()*, která při každé změně v IČ kopíruje do položky DIČ kód země (CZ) a obsah IČ. V případě potřeby je samozřejmě možné upravovat obsah DIČ.

V tomto okamžiku byl formulář hotový, takže bylo třeba zpracovat vstupy a ošetřit všechny možné stavy, jež by mohly nastat. To měl za úkol servlet, resp. třída Servlet, kterou nyní rozeberu.

4.4.2 Třída Servlet

V servletu jsem si nejprve uložil do proměnných typu String hodnoty všech parametrů z formuláře, abych je mohl ošetřit a dále zpracovávat. Prvním v pořadí je daňový doklad, u kterého se nejprve metodou *length()* kontroluje, jestli byly zadány čtyři znaky. Pokud byly, tak je daný řetězec převeden na celočíselné číslo metodou *parseInt()* třídy java.lang.Integer. Nelze-li řetězec převést nebo nemá-li dostatečnou délku, uloží se do atributu s názvem „dd“ hláška „Zadejte 4-místné číslo.“ a servlet přesměruje zpět na původní formulář. Tam je tento atribut vyžádán pomocí jazyka EL.

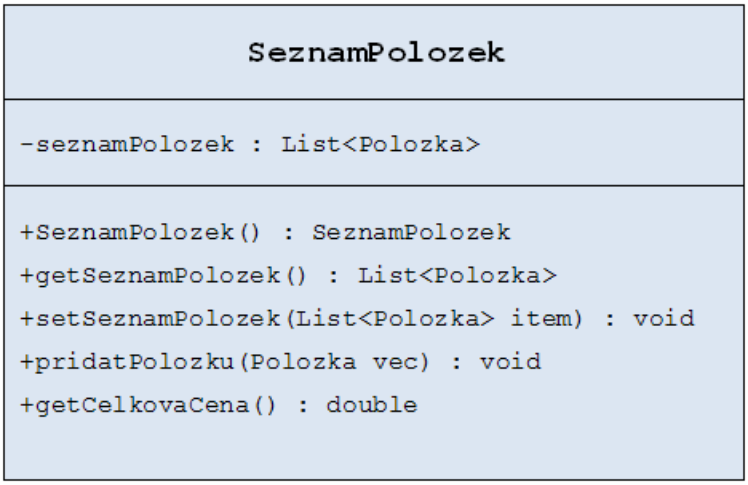
V sekci „Odběratel“ program pouze zkontroluje, zdali byly vyplněny všechny položky (kromě IČ a DIČ, ty rozeberu dále). V případě nějakého nevyplněného údaje se do atributu „odberatel“ nastaví text „Zadejte všechny údaje.“. Zde nemělo smysl to jakkoli více ošetřovat. Naproti tomu IČ a DIČ bylo nutné ošetřit a nebylo to tak jednoduché, jak jsem si zprvu myslel. V zásadě se jedná o to, že IČ může, ale také nemusí být vyplněno – v závislosti na tom, zda klient je či není

podnikatelským subjektem. Od toho se odvíjí DIČ[11], které rovněž může a nemusí být zadané podle toho, zda klient je či není plátce DPH. Navíc IČ nemusí mít vždy délku osmi číslic, jak je dnes standardní. Snažil jsem se proto zohlednit všechny možnosti, které by se mohly vyskytnout – tj. když zadavatel vyplní kolonku IČ, tak DIČ může být prázdné (tzn. klient je podnikatelský subjekt, ale není plátce DPH) a když vyplní DIČ, tak musí mít vyplněné také IČ (DIČ nemůže být bez IČ). Další možností je, že zákazník není podnikatelským subjektem (tím pádem nemá ani DIČ), tj. obě kolonky zůstanou prázdné a v tom případě program do atributů „ic“ a „dic“ nastaví pomlčku („–“).

Při odeslání formuláře se odešle mj. i parametr „vyber“, který má vždy hodnotu „1“ nebo „2“. Na základě této hodnoty se v servletu nastaví do atributu s názvem „doprava“ text „Zasláno na email odběratele“, příp. „Nakopírováno na server“.

Důležitou částí programu tvoří vyfakturované položky a jejich ceny. Vzhledem k tomu, že nelze dopředu určit, kolik takových položek bude (může jich být od jedné do pěti), je rozumné a vhodné je ukládat do seznamu. Jelikož je Java jazyk objektově orientovaný, vytvořil jsem si pro ukládání položek do seznamu třídu „SeznamPolozek“, jejíž proměnné a metody lze vidět na obrázku 4.5.

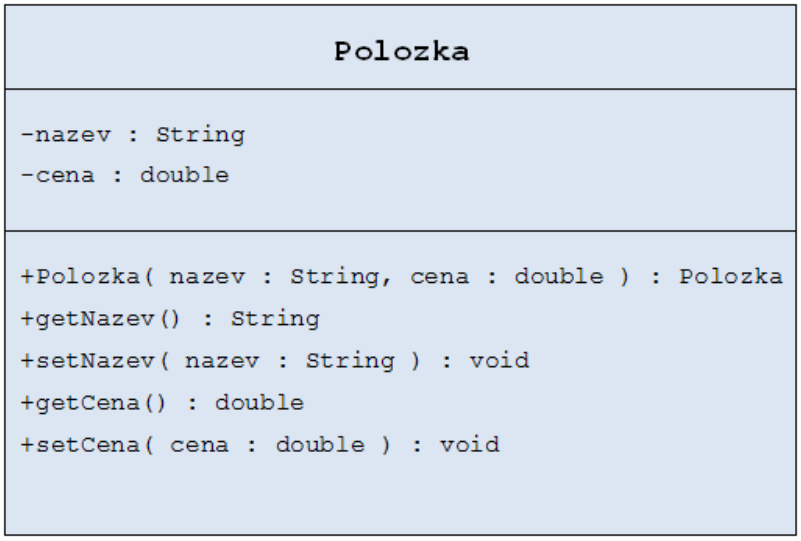
Obr. 4.4 Třída „SeznamPolozek“



[11] DIČ je u právnické osoby odvozeno od identifikačního čísla. U fyzické osoby je to pak rodné číslo, popř. jiný obecný identifikátor.

V třídě jsem deklaroval proměnnou „seznamPolozek“, která bude ukládat objekty (v tomto případě položky a ceny). Pro vytváření takových objektů jsem si vytvořil další třídu „Polozka“ (viz. obr. 4.5) a proměnnou seznamPolozek parametrizoval podle ní. Třída Polozka obsahuje dvě privátní proměnné – „nazev“ a „cena“.

Obr. 4.5 Třída „Polozka“



Jednotlivé položky a ceny jsem si ukládal do polí typu String s názvem „item“ pro položky a „price“ pro ceny. Poté servlet projede v cyklu každou položku a bude zjišťovat, jestli i-tá položka pole „item“ a stejně tak i-tá cena pole „price“ v sobě něco obsahují. V případě, že ano, servlet vytvoří novou instanci třídy „Polozka“ s i-tou položkou a i-tou cenou, kterou následně uloží do seznamu metodou *pridatPolozku()* třídy „SeznamPolozek“. Zohledněna byla také možnost zadání nesprávné hodnoty do pole s cenou neboli zadání alfabetického znaku. V tom případě se servlet vrátí zpět na form.jsp, zobrazí hlášku „Zadejte cenu ve správném formátu.“ a zároveň ukáže šipku vedle řádku, kde nastala chyba. Pokud by v i-tém řádku nebyla vyplněna cena, ale pouze název služby, příp. vice versa, tak servlet opět vrací požadavek zpět s hláškou „Zadejte název služby a její cenu.“ spolu s šipkou ukazující, kde se stala chyba, což ilustruje příloha č. 2 Chybové hlášky. Rovněž jsem ošetřil stav, kdyby žádná položka nebyla vyplněna. Je-li vše v pořádku, tj. stav, kdy je korektně vyplněná alespoň jedna položka s cenou, uloží

se vytvořený seznam s objekty do session pod jménem „seznam“. Nakonec se ještě uloží do pomocné proměnné „celkovaCenaZaokr“ součet všech prvků pole „price“, což se provede voláním metody *getCelkovaCena()*, která postupně prochází vytvořený seznam a vrací součet všech cen.

Zbývá již ošetřit poslední tři kolonky – záloha, sleva a DPH. U zálohy kontroluji, zda je to číslo a poté, jestli je to kladné číslo. Pokud by nastala chyba, tak se zobrazí formulář s příslušnou hláškou, viz příloha č. 2.

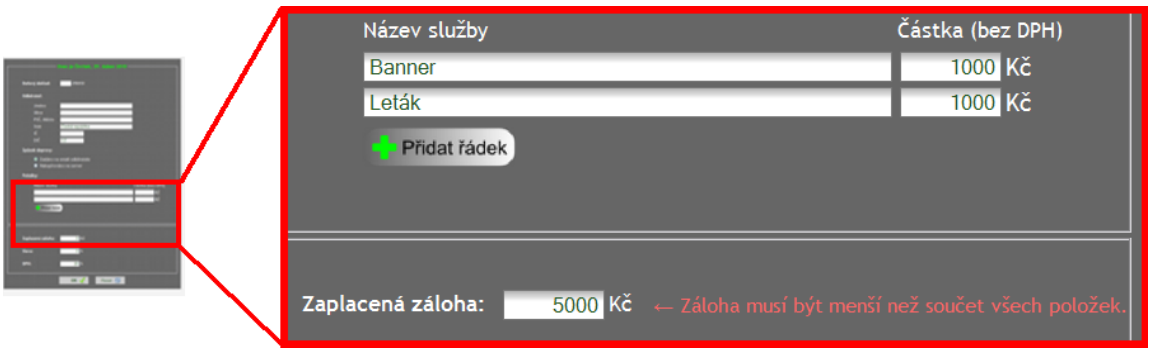
U slevy stačilo kontrolovat, jestli se nachází v rozmezí 0 až 1, resp. 0 až 100 % (nedá se sice předpokládat, že by někdy byla sleva 100%, ale pro jistotu jsem ponechal i tuto možnost). V každém jiném případě servlet opět vrací formulář s hláškou „Zadejte kladné číslo v rozmezí od 0 do 100.“, příp. „Zadejte slevu ve správném formátu.“ v závislosti na chybě, která nastala.

Posledním kontrolovaným prvkem je DPH. To se musí nalézat v rozmezí 1 až 99 %. Pro budoucí výpočty jsem jej dopředu vydělil 100 a přičetl 1, čili DPH může nabývat hodnot 1,01 až 1,99. Stejně jako u všech ostatních vstupů i zde jsem provedl opatření, která řeší všechny nežádoucí stavy, jak dokládá např. obrázek v příloze č. 2 Chybové hlášky.

Ošetřil jsem tedy všechny vstupy a stavy, jež by mohly nastat, ale bylo ještě třeba zajistit, aby servlet zobrazil výslednou stránku pouze v případě korektního vyplnění všech položek. Definoval jsem si proto na začátku proměnnou celočíselného typu (int) s názvem „vsechno_ok“, která se s každou korektně vyplněnou položkou ve formuláři inkrementuje. Na základě její hodnoty se servlet rozhodne, kam přesměruje požadavek. Jestliže tedy daňový doklad, údaje o odběrateli, IČ (DIČ se odvíjí od IČ, takže není třeba jej započítávat), záloha, sleva, DPH a nejméně jedna položka s názvem služby spolu s její cenou jsou vyplněny a úspěšně projdou kontrolou, tak může servlet přesměrovat na výslednou stránku. V rámci optimalizace aplikace jsem ještě přidal podmínku, která zjišťuje, jestli je záloha (částka vč. DPH) menší než celková cena všech položek (vynásobena sazbou

DPH). Pokud není, tak se zobrazí opět formulář s hláškou „Záloha musí být menší než součet všech položek.“, viz obr. 4.6.

Obr. 4.6 Ošetření výše zálohy



Pakliže i při této kontrole bude vše v pořádku, nebrání již nic tomu, aby servlet mohl přesměrovat požadavek na výslednou stránku s fakturou. Ovšem ještě předtím jsem musel provést dvě důležité věci – tou první bylo vypočítat a uložit si do atributu celkovou cenu s DPH zaokrouhlenou na celé koruny dolů (pro pozdější výpočty) metodou *random()* třídy java.lang.Math a tou druhou věcí pak nastavit a uložit si do atributu splatnost, tj. přičíst k aktuálnímu datu 7 dní. Výše popsané vypadá v praxi následovně:

```
request.setAttribute( "celkovaCenaSdphZaokr" , Math.floor( ( celkovaCenaZaokr –
      – ( celkovaCenaZaokr * sleva ) ) * dph ) );
Calendar kalendar = Calendar.getInstance();   // tímto jsem vytvořil novou instanci kalendáře
kalendar.add( Calendar.DAY_OF_MONTH , 7 ); // metodou add() přidám k akt. datu 7 dní
request.setAttribute( "splatnost" , new SimpleDateFormat( "d.M.yyyy" ).format(
      kalendar.getTime() ) );   // vytvořené datum naformátuji dle zadaného vzoru
```

V tomto bodě je již vše připraveno k zobrazení konečné faktury.

4.4.3 Výsledná faktura

Pro zobrazení výstupu aplikace jsem si v projektu vytvořil novou stránku s názvem **result.jsp**, jejíž vzhled by měl přesně odpovídat tomu, jaký je v příloze č. 1

Původní faktura. Nabízela se také možnost nevytvářet novou stránku, ale pouze vložit kód do té původní – form.jsp, a to s použitím např. standardních JSTL značek. V JSP stránce bych si definoval jednoduchou podmínku, jestliže servlet uloží do atributu (např. s názvem „form“) hodnotu „true“, tak se vykoná část kódu zobrazující formulář, v případě opačném se provede část druhá, která řeší výstup programu. Praktická ukázka kódu v JSP stránce:

```
<c:choose>                          // „choose“ plní stejnou funkci jako přepínač „switch“
    <c:when test=" ${ form } " >    // „when“ je obdoba „case“
        …  <form> </form>  …        // v případě, že ve „form“ je true, zobrazí se formulář
    </c:when>
    <c:otherwise>                   // pokud „form“ nebude true, vykoná se tělo této značky
        …  (výstup programu)  …
    </c:otherwise>
</c:choose>
```

Ale z hlediska přehlednosti a také kvůli kaskádovým stylům jsem se rozhodl udělat dvě separátní stránky.

Nejprve jsem si vytvořil základní layout (neboli rozvržení) stránky a snažil se nastylovat ji tak, aby byla maximálně podobná s fakturou od zadavatele. To s sebou neslo mnohá úskalí, jelikož každý prohlížeč, ve kterém jsem aplikaci testoval, zobrazoval výsledek trochu jinak než ostatní. I když se jednalo v zásadě o detaily, tak bylo třeba zajistit jednotný vzhled ve všech prohlížečích, aby výsledný PDF soubor vypadal vždy stejně. Navíc uživatel nechce být limitován možností použít pouze určitý prohlížeč, ve kterém by se aplikace zobrazovala korektně. Proto jsem po diskuzi se zadavatelem optimalizoval stránky na prohlížeče Mozilla Firefox, Apple Safari a Opera. Optimalizace mi zabrala velkou část času při realizaci projektu, ale nakonec jsem docílil prakticky identického vzhledu ve všech třech výše zmiňovaných prohlížečích.

Poté jsem se mohl věnovat programové složce. V servletu jsem si ukládal vstupní data a různé propočty do atributů, abych si je mohl na této stránce vyzvednout

a použít. Jako první jsem potřeboval vytvořit 10místné číslo faktury, které se skládá ze 4místného uživatelem zadaného čísla a zbylých 6 číslic je odvozeno od aktuálního data. Obě hodnoty jsou uloženy v atributech „doklad“ a „date“, takže stačilo je vyzvednout pomocí jazyka EL a dát vedle sebe. Datum bylo nutné ještě pomocí JSTL značek „přeparsovat“ do určitého formátu, uložit do proměnné a tu zformátovat dle vzoru „ddMMyy“. Stejný postup jsem aplikoval i v sekci „Dodavatel“ u variabilního symbolu. Zbývající údaje v této sekci jsou vždy stejné. Dále bylo nutné vyplnit údaje o odběrateli, datum vystavení, splatnost atd. Všechny potřebné informace byly uloženy v atributech, takže stačilo si je pouze vyžádat.

Pro výpis fakturovaných položek jsem použil tabulku, jejíž každý řádek obsahuje název položky, procentní sazbu DPH, částku bez DPH, DPH vypočtené z ceny položky a konečnou částku vč. DPH. Všechny vyplněné položky s cenami jsou uloženy v kolekci typu ArrayList parametrizované dle třídy Polozka, která byla uložena v servletu do atributu s názvem „seznam“. Tento atribut je typu SeznamPolozek, tudíž na něm lze volat metodu *getSeznamPolozek()* vracející seznam položek, resp. kolekci typu List opět parametrizovanou dle třídy Polozka. Program v cyklu prochází každý prvek kolekce a ukládá jej postupně do proměnné „vec“. Na této proměnné se k získání příslušného názvu položky a ceny v každém cyklu volají metody třídy Polozka *getNazev()* a *getCena()*. Dále bylo ještě třeba provést příslušné propočty u každé položky a výsledek dle zadaného vzoru naformátovat. Z praktické ukázky bude výše řečené jasnější:

```
<table>
    <tr><th>Název</th> <th>DPH %</th> <th>Základ</th> <th>DPH</th> <th>Celkem </th> </tr>
    <c:forEach items=" ${ seznam.seznamPolozek } "  var="vec">
        <tr><td> ${ vec.nazev } </td> <td> ${ dph } % </td>
            <td><fmt:formatNumber pattern="#,##0.00"> ${ vec.cena - ( vec.cena * sleva ) }
                </fmt:formatNumber> </td>
            <td><fmt:formatNumber pattern="#,##0.00"> ${ ( vec.cena - ( vec.cena * sleva ) )
                div 100 * dph } </fmt:formatNumber> </td>
            <td><fmt:formatNumber pattern="#,##0.00"> ${ ( vec.cena - ( vec.cena * sleva ) ) *
                (dph div 100 + 1 ) } </fmt:formatNumber> </td> </tr>
```

```
    </c:forEach>
</table>
```

Nakonec bych zmínil už pouze tabulku s rekapitulací cen, jelikož ostatní údaje jako záloha apod. jsou jen otázkou vyžádání si příslušného atributu. Sumarizační tabulka je velmi podobná té předchozí, pouze s tím rozdílem, že na proměnné „vec“ se navíc volá metoda *getCelkovaCena()*, kterou jsem výše popisoval. Ta našla uplatnění zejména při sčítání celkových položek.

Aplikace byla tedy hotová a zbývalo už jen převést stránku result.jsp do formátu PDF skrz nainstalovanou virtuální tiskárnu. V rámci vylepšování aplikace jsem ještě přidal JavaScriptovou funkci, která při stisku klávesy mezerník vyvolá dialogové okno pro tisk. Tam již stačí vybrat požadovanou tiskárnu „Adobe PDF“ a stránka se zkonvertuje. Ovšem hned při prvním pokusu jsem narazil na několik problémů. Tím nejmarkantnějším bylo, že se převedl pouze holý text, tj. bez grafických prvků. V zápětí bylo jasné, že se sice vyexportovala výsledná stránka, avšak bez přiloženého CSS souboru. Na vině byla absence atributu „media“ v elementu `<link />`, kterým jsem linkoval (připojoval) kaskádové styly. Stačilo v něm doplnit daný atribut s parametrem „print“ a export již poté proběhl v pořádku.

```
<link href="result.css" rel="stylesheet" type="text/css" media="screen, print" />
```

Poslední překážkou už bylo jenom vypnout v prohlížeči v nastavení tiskárny možnost tisku záhlaví, zápatí a příp. i dalších věcí, které do faktury nepatří. Vyzkoušel jsem export v několika prohlížečích a překvapilo mne, že v každém byly výsledky mírně odlišné (v případě Internet Exploreru i velmi odlišné). Jednoznačně nejlépe vycházel export v Opeře a ve Firefoxu viz příloha č. 3, resp. 4, takže zadavateli doporučím ad hoc používat právě tyto dva prohlížeče.

V tomto okamžiku byl program kompletně hotov, otestován a připraven k implementaci u zadavatele.

4.5 Implementace aplikace u zadavatele

Realizovaný program jsem celou dobu vyvíjel a testoval v prostředí NetBeans, které se staralo o spouštění a zastavování Tomcatu. Z hlediska uživatelského komfortu to není příliš efektivní způsob spouštění aplikace – počkat než se spustí NetBeans, než nastartuje Tomcat a případně prohlížeč. Hledal jsem proto řešení, jak co nejefektivněji vyřešit tento problém. Uživatel chce vždy klikat co nejméně a mít vše rychle, takže pro něj by bylo ideální řešení mít na ploše ikonu (nebo zástupce), na kterou když pokliká, tak se mu spustí prohlížeč s danou aplikací. Jediným reálným řešením tedy bylo vytvořit dávkový soubor, který by nastartoval Tomcat a zároveň spustil prohlížeč s URL adresou projektu. Podařilo se mi najít skript, kterým se dá spustit Tomcat (v mém případě trvá jeho start 1,5 sekundy). Skript jsem si upravil a doplnil ho o řádek na spuštění prohlížeče Safari s adresou projektu „http://localhost:8081/Faktura/“. Celý skript vypadá následovně:

```
start safari.exe "http://localhost:8081/Faktura/"
java -Djava.endorsed.dirs="..\common\endorsed" -classpath "c:\Program
Files\Java\jdk1.6.0_10\lib\tools.jar;..\bin\bootstrap.jar" -Dcatalina.base=".."
-Dcatalina.home=".." -Djava.io.tmpdir="..\temp" org.apache.catalina.startup.Bootstrap start
```

Vše fungovalo výborně, ovšem na platformě Windows. Původně jsem se domníval, že na Mac OS to bude podobné a že nebude problém docílit stejného výsledku, tj. hlavně spustit Tomcat přes příkazový řádek, resp. Terminál[12]. Ovšem záhy se ukázalo, že to bude větší problém, než jsem si představoval. Nejprve jsem u zadavatele stáhnul a nainstaloval Tomcat a zkusil výše popsaný způsob. Ač jsem předpokládal, že bude fungovat, nestalo se tak a musel jsem zkusit najít nějaké řešení na Internetu. Našel jsem pouze dva relevantní zdroje, které sice nevyřešily tento problém, ale aspoň z části mi pomohly osvětlit danou problematiku. S platformou Mac OS jsem nikdy předtím neměl možnost se blíže seznámit a zadavatel se s podobnou záležitostí také nikdy nesetkal, takže jsme zkoušeli

[12] Terminál je obdoba příkazového řádku ve Windows.

metodou „pokus-omyl“. Po asi dvou konzultacích a mnoha pokusech jsme přišli na to, jak alespoň spustit Tomcat přes Terminál. Nakonec stačilo zadat příkaz:

```
$cd ~/Desktop/apache-tomcat-6.0.26/bin/startup.sh
```

Pro zastavení serveru stačí místo „startup.sh“ zadat „shutdown.sh“. Tím se vyřešila jedna část – spustit Tomcat. Další a neméně důležitou částí bylo přijít na to, jak vytvořit dávkový soubor, kterým bychom dosáhli spuštění prohlížeče s požadovaným URL a který by bylo možno umístit na plochu (a jednoduše spouštět). Opět proběhla konzultace se zadavatelem a hledání řešení na Internetu. Výsledkem bylo napsat následující AppleScript a uložit jej jako aplikaci (v Mac OS se dají spouštět aplikace pouze s příponou „app“):

```
tell application "Terminal"
    do script "$cd ~/Desktop/apache-tomcat-6.0.26/bin/startup.sh"
    do script "open http://localhost:8080/Faktura"
end tell
```

Uložený soubor (aplikaci) lze umístit na plochu a poklikáním spouštět – což je přesně stav, kterého jsme chtěli se zadavatelem dosáhnout.

Posledním krokem pak bylo provést „deploy“ neboli nasazení aplikace na server. Možností je hned několik, já si vybral tu nejjednodušší – nakopíroval jsem celý projekt do složky, kde je nainstalován Tomcat, konkrétně do *\webapps*. Tím bylo zaručeno, že jej Tomcat nahraje.

5 Zhodnocení navrhovaného řešení

Projekt byl obecně navržen jako webová aplikace, která zobrazí výslednou fakturu jako HTML stránku a tu poté uživatel vytiskne jako PDF dokument pomocí příslušného softwaru. Zhodnotit navrhované řešení se dá z hlediska jeho kladů a záporů. Mezi hlavní klady bych zařadil to, že takové řešení je:

- *univerzální*, ve smyslu platformě nezávislé (nezáleží tedy na operačním systému, výsledek bude stejný, ať už uživatel používá platformu Windows nebo třeba Mac OS),
- *efektivní* (uživatel vyplní pouze nejnutnější položky a program obstará vše za něj – kontrola vstupních dat, grafický výstup),
- *finančně nenákladné* – nejsou zde v podstatě žádné přímé náklady spojené s realizací řešení.

Za jistou nevýhodu se dá považovat repetitivní zadávání stejných údajů (myšleno identifikační údaje o klientovi, kterému již byla dříve vystavena faktura), jelikož aplikace si je nikde neuchovává. Jedná se sice jenom o pár údajů (proto to zadavatel neuváděl v požadavcích), ale přispělo by to ještě více k uživatelskému komfortu.

Do budoucna se tedy nabízí rozšíření aplikace o spojení s databází, konkrétně s MySQL, kterou v současné době zadavatel využívá. Pro uživatele se téměř nic nezmění, z jeho pohledu bude vše tak, jak byl zvyklý – ve vstupním formuláři vyplní všechny položky, následně ho odešle a v případě korektního vyplnění se mu zobrazí výsledná faktura, kterou si převede do PDF. Rozdíl ovšem bude v tom, že ještě předtím, než servlet přesměruje požadavek na výslednou stránku, se zavolá procedura, která zadaná data uloží do databáze – zejména informace o klientovi (jako je např. jméno, adresa, IČ), ale i název provedené služby a její cenu. V úvodním formuláři by proto přibyl u jména jeden select box, ze kterého by si uživatel mohl vybrat jméno zákazníka a tím by se mu vyplnily všechny o něm

uložené údaje do příslušných kolonek. Samozřejmě za podmínky, že danému zákazníkovi již byla někdy nějaká faktura vystavena a tím pádem uložena do databáze. Odpadlo by tím zbytečné vypisování již dříve zadaných údajů a také možnost chyby při ručním zadávání. Další možnost je přidat na úvodním formuláři textové pole pro poznámku, která by se ukládala do databáze spolu s ostatními údaji. Díky záznamům uložených v databázi by bylo možné rovněž efektivně vyhledávat informace o každém klientovi, a to i dle různých parametrů – např. zobraz první tři faktury s nejvyšší vyfakturovanou částkou nebo všechny klienty, kteří jsou z Nového Jičína apod.

Dalším možným rozšířením by mohlo být doplnění možnosti editace údajů o zadavateli, např. telefonní číslo, adresa, apod. Tyto údaje se nebudou pravděpodobně často měnit, ale z hlediska maximální funkčnosti programu by tato možnost mohla být zohledněna.

6 Závěr

V rámci své práce jsem nejprve představil zadavatele, firmu Graphic4web, popsal jeho dosavadní způsob zasílání vystavovaných faktur zákazníkům a rovněž uvedl i důvody, proč hledal jiné, lepší řešení. Dále jsem stručně popsal všechny důležité technologie a programy, jež byly použity při realizaci projektu. Mezi ty stěžejní technologie použité při realizaci projektu se řadí Java spolu s XHTML a mezi hlavní programy pak NetBeans a Adobe Acrobat. Na základě požadavků zadavatele jsem vytvořil webovou aplikaci založenou zejména na kombinaci technologií servletů a JSP stránek, jejíž výstup bude vyexportován do souboru PDF. Poté jsem zhodnotil přínos takového řešení a nastínil možnost budoucího rozvoje aplikace, kterou, jak předpokládám, budu v budoucnu ještě realizovat.

Cílem mé bakalářské práce bylo navrhnout a vytvořit efektivní, uživatelsky přívětivý program na tvorbu faktur, který by nahradil stávající řešení zadavatele. Vytyčený cíl se mi podařilo splnit – vytvořená aplikace splňuje všechny požadavky zadavatele. Ten je s výsledným řešením velmi spokojen a v současné době ho již využívá.

Díky tomuto projektu jsem si osvojil práci s různými technologiemi, resp. programovacími a značkovacími jazyky, jako např. XHTML, JavaScript, Java servlety, JSP stránky, JSTL značky, Expression Language ad. V současné době je velká poptávka po webových aplikacích a tento trend neustále roste, takže jsem přesvědčen, že vědomosti a zkušenosti získané v průběhu této práce mi budou přínosem a zároveň impulsem k tomu, abych se začal věnovat této perspektivní oblasti ICT.

Seznam použité literatury

Publikace:

1. BURD, B., *JSP: JavaServer Pages – Podrobný průvodce.* 1. vyd. Praha: Computer Press, 2003. 381 s. ISBN 80-7226-804-X.
2. CYROŇ, M., *CSS – kaskádové styly : praktický manuál.* 1. vyd. Praha: Grada Publishing, 2006. 340 s. ISBN 80-247-1420-5.
3. HALL, M., *Java servlety a stránky JSP*. 1. vyd. Praha: Neocortex, 2002. 574 s. ISBN 80-86330-06-0.
4. HLAVENKA, J., et al. *Vytváříme WWW stránky*. 4. vyd. Praha: Computer Press, 2000. 520 s. ISBN 80-7226-293-9.
5. Sylaby předmětu Programování – vyučující Ing. V. Novák, Ph.D.

Internetové zdroje:

6. Adobe Acrobat [online]. [cit. 11.4.2010]. Dostupné na WWW: <http://cs.wikipedia.org/wiki/Adobe_Acrobat>.
7. Apache Tomcat [online]. [cit. 11.4.2010]. Dostupné na WWW: <http://en.wikipedia.org/wiki/Apache_Tomcat>.
8. CDDL a GNU General Public License [online]. [cit. 9.4.2010]. Dostupné na WWW: <http://netbeans.org/cddl-gplv2.html>.
9. Expression Language [online]. [cit. 11.4.2010]. Dostupné na WWW: <http://en.wikipedia.org/wiki/Expression_Language>.
10. Faktura [online]. [cit. 12.4.2010]. Dostupné na WWW: <http://www.euroekonom.cz/podnikani-faktura.php>.
11. Graphic4web [online]. [cit. 8.4.2010]. Dostupné na WWW: <http://www.graphic4web.com>.
12. HyperText Markup Language [online]. [cit. 9.4.2010]. Dostupné na WWW: <http://cs.wikipedia.org/wiki/HyperText_Markup_Language>.
13. JavaServer Pages pro všechny [online]. [cit. 11.4.2010]. Dostupné na WWW: <http://interval.cz/clanky/javaserver-pages-pro-vsechny>.

14. Java Servlets – predstavenie technologie [online]. [cit. 11.4.2010]. Dostupné na WWW: <http://interval.cz/clanky/java-servlets-predstavenie-technologie>.
15. Microsoft Office Visio 2007 [online]. [cit. 11.4.2010]. Dostupné na WWW: <http://www.microsoft.com/cze/office/programs/visio/highlights.mspx>.
16. Proforma faktura [online]. [cit. 12.4.2010]. Dostupné na WWW: <http://cs.wikipedia.org/wiki/Proforma_faktura>.
17. Vývojové prostředí [online]. [cit. 9.4.2010]. Dostupné na WWW: <http://cs.wikipedia.org/wiki/Vývojové_prostřed%C3%AD>.
18. XHTML [online]. [cit. 11.4.2010]. Dostupné na WWW: <http://cs.wikipedia.org/wiki/XHTML>.
19. XHTML – základní struktura dokumentu [online]. [cit. 10.4.2010]. Dostupné na WWW: <http://interval.cz/clanky/xhtml-zakladni-struktura-dokumentu>.

Seznam zkratek

(X)HTML – (eXtensible) HyperText Markup Language
ad hoc – z latiny, překládá se jako „za určitým účelem“ nebo „pro tento konkrétní případ“
ad. – a další
AJAX – Asynchronous JavaScript and XML
angl. – anglicky
apod. – a podobně
atd. – a tak dále
CDDL – Common Development and Distribution License
CSS – Cascading Style Sheets
č. – číslo
ddMMyy – datum ve formátu den v měsíci (s vodící nulou) , měsíc v roce (s vodicí nulou) a rok (s vodící nulou)
DIČ – daňové identifikační číslo
DPH – daň z přidané hodnoty
GNU General Public License – všeobecná veřejná licence GNU
GUI – Graphical User Interface
HTTP – HyperText Transfer Protocol
ICT - Information and Communication Technologies
IČ – identifikační číslo
IDE – Integrated Development Environment
J2EE – Java Enterprise Edition
J2ME – Java Micro Edition
J2SE – Java Standard Edition
JPG (JPEG) – Joint Photographic Experts Group
JRE – Java Runtime Environment
JSTL – JSP Standard Tag Library
JVM – Java Virtual Machine
mj. – mimo jiné

MySQL – My Structured Query Language
např. – například
obr. – obrázek
OS – operační systém
PHP – Hypertext Preprocessor
PNG – Portable Network Graphic(s)
popř. – popřípadě
příp. – případně
PSČ – poštovní směrovací číslo
resp. – respektive
SEO – Search Engine Optimization
str. – strana
tj. – to jest
tzn. – to znamená
tzv. – tak zvaný / tak zvaně
UK – Univezita Karlova
UML – Unified Modeling Language
URL – Uniform Ressource Locator
vč. – včetně
vice versa – naopak, obráceně
viz – videlicet (lze vidět)
W3C – World Wide Web Consortium
WWW – World Wide Web
WYSIWYG – What You See Is What You Get
XML – eXtensible Markup Language

Prohlášení o využití výsledků bakalářské práce

Prohlašuji, že

- byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. – autorský zákon, zejména § 35 – užití díla v rámci občanských a náboženských obřadů, v rámci školních představení a užití díla školního a § 60 – školní dílo;
- beru na vědomí, že Vysoká škola báňská – Technická univerzita Ostrava (dále jen VŠB-TUO) má právo nevýdělečně ke své vnitřní potřebě bakalářskou práci užít (§ 35 odst. 3);
- souhlasím s tím, že jeden výtisk bakalářské práce bude v elektronické podobě archivován v Ústřední knihovně VŠB-TUO a jeden výtisk bude uložen u vedoucího bakalářské práce. Souhlasím s tím, že bibliografické údaje o bakalářské práci, budou zveřejněny v informačním systému VŠB-TUO;
- bylo sjednáno, že s VŠB-TUO, v případě zájmu z její strany, uzavřu licenční smlouvu s oprávněním užít dílo v rozsahu § 12 odst. 4 autorského zákona;
- bylo sjednáno, že užít své dílo, bakalářskou práci, nebo poskytnout licenci k jejímu využití mohu jen se souhlasem VŠB-TUO, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly VŠB-TUO na vytvoření díla vynaloženy (až do jejich skutečné výše).

V Ostravě dne ……..

……..

jméno a příjmení studenta

Adresa trvalého pobytu studenta:

Na Lani 222
Nový Jičín - Loučka
741 03

Seznam příloh

Příloha č. 1 Původní faktura

FAKTURA - DAŇOVÝ DOKLAD 1111040310

Dodavatel:

Richard ŽIŽKA
Palackého 1159/114
741 01 NOVÝ JIČÍN
Česká republika

graphic web

IČ: 732 14 175
DIČ: CZ8505255704
Sídlo: Palackého 1159/114, 741 01 NOVÝ JIČÍN, Česká republika
Telefon: +420 777 991 126
E-mail: info@graphic4web.com
Internet: www.graphic4web.com
Účet: **KB Nový Jičín, 86-6381840237/0100, v.s.: 1111040310, k.s.: 0308**

Datum vystavení: 04. 03. 2010
Datum uskut. zdaň. plnění: 04. 03. 2010
Splatnost: **11. 03. 2010**
Způsob platby: Převodem
Způsob dodání: Elektronicky
Vystavil: Richard ŽIŽKA

Odběratel / fakturační adresa:

Jan Novák
Masarykova 5
123 45 VSETÍN
Česká republika

IČ: 123 45 678
DIČ: CZ12345678

Jan Novák
Masarykova 5
123 45 VSETÍN
Česká republika

Název služby	DPH	Základ	DPH	Celkem
Banner	20%	1 007,00	201,40	1 208,40

Zaplacená záloha **500,00**

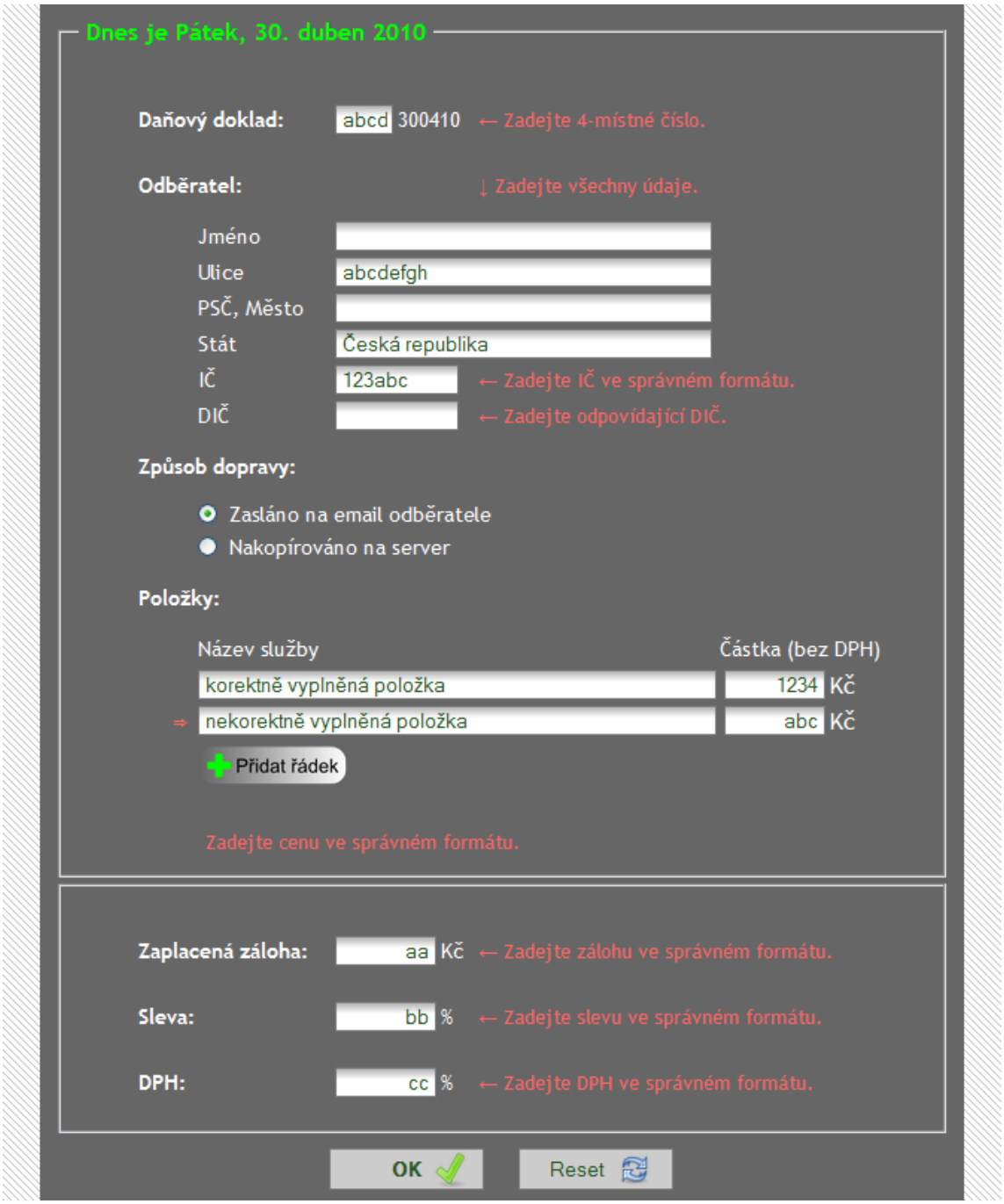
Celková hodnota faktury v Kč: **708,00**
Již zaplaceno: 500,00
Celkem k úhradě 1 208,00

DPH	Základ	DPH	Celkem
20%	590,33	118,07	708,40
	590,33	118,07	708,40
Zaokrouhlení			-0,40

Převzal _____ **dne** _____

graphic web
tvorba a vývoj profesionálních www prezentací
Richard Žižka
Palackého 114, 741 01 **Nový Jičín**
IČO: 73214175 DIČ: **374-8505255704**
Telefon: 00420 **723 789 668**

Příloha č. 2 Chybové hlášky



Příloha č. 3 Výsledná faktura vyexportovaná z Opery

FAKTURA - DAŇOVÝ DOKLAD 1111050510

Dodavatel:

Richard ŽIŽKA
Palackého 1159/114
741 01 NOVÝ JIČÍN
Česká republika

graphic4web

IČ: 732 14 175
%0l: CZ8505255704
Sídlo: Palackého 1159/114, 741 01 NOVÝ JIČÍN, Česká republika
Telefon: +420 777 991 126
E-mail: info@graphic4web.com
Internet: http://www.graphic4web.com/
Účet: **KB Nový Jičín, 86-6381840237/0100, v.s.: 1111050510, k.s.: 0308**

Datum vystavení: 05. 05. 2010
Datum uskut. zdaň. plnění: 05. 05. 2010
Splatnost: **12. 05. 2010**
Způsob platby: Převodem
Způsob dopravy: Zasláno na email odběratele
Vystavil: Richard ŽIŽKA

Odběratel / fakturační adresa:

Jan Novák
Masarykova 5
123 45 VSETÍN
Česká republika

IČ: 12345678
DIČ: CZ12345678

Jan Novák
Masarykova 5
123 45 VSETÍN
Česká republika

Název služby	DPH %	Základ	DPH	Celkem
První položka (Opera)	20%	1 000,00	200,00	1 200,00
Druhá položka (Opera)	20%	1 234 567,00	246 913,40	1 481 480,40

Zaplacená záloha 10 000,00

Celková hodnota faktury v Kč: **1 472 680,00**
192zaplaceno: 10 000,00
Celkem k úhradě: 1 482 680,00

DPH %	Základ	DPH	Celkem
20%	1 000,00	200,00	1 200,00
20%	1 234 567,00	246 913,40	1 481 480,40
	1 235 567,00	247 113,40	1 482 680,40
Zaokr.:			-0,40

Převzal ..... **dne** .....

graphic 4 web
tvorba a vývoj profesionálních www prezentací
Richard Žižka
Palackého 114, 741 01 Nový Jičín
IČO: 73214175 DIČ: 374-8505255704
Telefon: 00420 723 789 668

Příloha č. 4 Výsledná faktura vyexportovaná z Firefoxu

FAKTURA - DAŇOVÝ DOKLAD 1111050510

Dodavatel:

Richard ŽIŽKA
Palackého 1159/114
741 01 NOVÝ JIČÍN
Česká republika

graphic4web

IČ: 732 14 175
DIČ: CZ8505255704
Sídlo: Palackého 1159/114, 741 01 NOVÝ JIČÍN, Česká republika
Telefon: +420 777 991 126
E-mail: info@graphic4web.com
Internet: http://www.graphic4web.com/
Účet: KB Nový Jičín, 86-6381840237/0100, v.s.: 1111050510, k.s.: 0308

Datum vystavení: 05. 05. 2010
Datum uskut. zdaň. plnění: 05. 05. 2010
Splatnost: **12. 05. 2010**
Způsob platby: Převodem
Způsob dopravy: Zasláno na email odběratele
Vystavil: Richard ŽIŽKA

Odběratel / fakturační adresa:

Jan Novák
Masarykova 5
123 45 VSETÍN
Česká republika

IČ: 12345678
DIČ: CZ12345678

Jan Novák
Masarykova 5
123 45 VSETÍN
Česká republika

Název služby	DPH %	Základ	DPH	Celkem
První položka	20%	1 000,00	200,00	1 200,00
Druhá položka	20%	1 234 567,00	246 913,40	1 481 480,40

Zaplacená záloha 10 000,00

Celková hodnota faktury v Kč: **1 472 680,00**
Již zaplaceno: 10 000,00
Celkem k úhradě: 1 482 680,00

DPH %	Základ	DPH	Celkem
20%	1 000,00	200,00	1 200,00
20%	1 234 567,00	246 913,40	1 481 480,40
	1 235 567,00	247 113,40	1 482 680,40
Zaokr.:			-0,40

Převzal _____ **dne** _____

graphic web
tvorba a vývoj profesionálních www prezentací
Richard Žižka
Palackého 114, 741 01 Nový Jičín
IČO: 73214175 DIČ: 374-8505265704
Telefon: 00420 723 789 668